Alexis Balayre

Future Position Prediction for Pressure Refuelling Port of Commercial Aircraft

School of Aerospace, Transport and Manufacturing
Computational and Software Techniques in Engineering

MSc
Academic Year: 2023–2024

Supervisors: Dr Boyu Kuang and Dr Stuart Barnes
May 2024

School of Aerospace, Transport and Manufacturing
Computational and Software Techniques in Engineering

MSc

Academic Year: 2023–2024

Alexis Balayre

Future Position Prediction for Pressure Refuelling Port of Commercial Aircraft

Supervisors: Dr Boyu Kuang and Dr Stuart Barnes
May 2024

This thesis is submitted in partial fulfilment of the requirements for the degree of MSc.

Academic Integrity Declaration

I declare that:

- the thesis submitted has been written by me alone.
- the thesis submitted has not been previously submitted to this university or any other.
- that all content, including primary and/or secondary data, is true to the best of my knowledge.
- that all quotations and references have been duly acknowledged according to the requirements of academic research.

I understand that to knowingly submit work in violation of the above statement will be considered by examiners as academic misconduct.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ML	Machine Learning
DL	Deep Learning
AI	Artificial Intelligence
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
EKF	Extended Kalman Filter
AAGR	Autonomous Aircraft Ground Refueling
AGR	Aircraft Ground Refueling
UAV	Unmanned Aerial Vehicle
AAR	Autonomous Aerial Refueling
DGPS	Differential Global Positioning System
SVM	Support Vector Machine
HOG	Histogram of Oriented Gradients
SOTA	State-of-the-Art
AIS	Automatic Identification System
GPS	Global Positioning System

Chapter 1

Introduction

Ground pressure refuelling is a standard method used to refuel commercial aircraft safely and efficiently. This process involves using a hydrant system, which consists of underground fuel pipelines connected to a network of fuel hydrants located at aircraft parking positions [**?**]. The hydrant system is supplied with fuel from storage tanks, typically located near the airport [**?**].

When an aircraft is ready for refueling, a hydrant dispenser vehicle, also known as a hydrant truck or cart, is connected to the hydrant pit using a flexible hose [**?**]. The hydrant dispenser vehicle is equipped with a pressure control valve, a flow meter, and a filtration system to ensure that the fuel meets the required quality standards [**?**].

The refueling process begins by connecting the hydrant dispenser vehicle to the aircraft's fuel panel using another flexible hose [**?**]. The pressure control valve on the hydrant dispenser vehicle is then used to regulate the fuel pressure and flow rate, ensuring that the fuel is delivered to the aircraft at the appropriate pressure and volume [**?**].

One of the main advantages of pressure ground refueling is its efficiency. This method allows for high fuel flow rates, which can significantly reduce aircraft turnaround times [**?**]. Additionally, the use of underground pipelines eliminates the need for fuel trucks, reducing traffic congestion and the risk of accidents on the apron [**?**].

Safety is another critical aspect of pressure ground refueling. The hydrant system is designed with multiple safety features, such as emergency shutdown valves and leak detection systems, to minimise the risk of fuel spills and fires [**?**]. Moreover, the hydrant dispenser vehicles are equipped with safety devices, such as dead man switches and bonding cables, to prevent incidents during the refueling process [**?**].

Figure 1.1: Pressure Refuelling of a Commercial Aircraft. Source: Tom Boon/Simple Flying

The aviation industry is undergoing a significant transformation with the advent of intelligent airports based on highly automated systems. Among these, automated refuelling systems play a crucial role in ensuring efficient and accurate refuelling of aircraft. However, one of the main challenges of this automation process is the accurate detection of the aircraft's refuelling port, which is relatively small and can easily be obscured by other visual elements on or near an aircraft. Scanning the entire area of each video frame is both time-consuming and inaccurate. It is therefore essential to develop a more efficient and accurate method of locating the refuelling port.

This thesis aims to address this challenge by developing a new AI model that uses the temporal relationships between successive frames of a video to predict the location of the refuelling port in subsequent frames. By focusing the analysis on the most relevant areas of the video sequence, this approach has the potential to optimise both the speed and accuracy of the refuelling system.

Specific objectives of this thesis include conducting a comprehensive review of state-of-the-art object detection and tracking methods, designing and developing a real-time computer vision system capable of accurately detecting and tracking the pressurised refuelling port of a commercial aircraft, the implementation and evaluation of deep learning time series models for future position prediction, the integration of Extended Kalman Filtering (EKF) into deep learning models to improve the accuracy and robustness of future position predictions, and the development of a real-time framework for predicting the future position of the pressurised refuelling port.

By achieving these objectives, this thesis aims to make a significant contribution to the development of intelligent airport systems and to improve the efficiency and accuracy of automated refuelling systems at airports, while reducing computing power requirements. The proposed framework has the potential to be applied in various scenarios, such as different lighting conditions, angles and orientations of refuelling ports, making it a versatile and effective solution to the challenges of automated refuelling systems.

Chapter 2

Literature Review

2.1 Automated Refulling Systems in the Aviation Industry

Automated refueling systems have gained substantial importance in the aviation industry due to their potential to enhance safety and efficiency. The concept of Autonomous Aircraft Ground Refueling (AAGR) emerged in the 1980s, with initial implementations featuring numbered markers near refueling ports to aid in image processing and robotic automation [**?** **?** **?**].

These systems streamline the refuelling process by using state-of-the-art mechanisms and technologies to ensure accuracy and speed. A key element is the pressurised fuel adaptor, which connects seamlessly to the aircraft [**?**]. Methods such as 'PosEst' have been proposed to capture and track objects in 3D, enabling the pressurised refuelling port to be located precisely. [**?**].

In addition, autonomous in-flight refuelling technologies for Unmanned Aerial Vehicles (UAV-AARs) have also been developed to extend their range, endurance and payload capacity without significantly altering their original design. This technology overcomes limitations in fuel capacity and endurance, enabling UAVs to undertake longer missions [**?**].

The implementation of AAGR presents a number of challenges, including varying lighting conditions, different refuelling port designs and potential obstructions. Confidentiality issues associated with aircraft refuelling data, as well as the lack of standardised workflows, also complicate the deployment of advanced AAGR solutions, based on [**?**] data.

Accurately detecting and estimating the position of the fuelling adaptor is a major hurdle. Current methods that rely on artificial features often encounter occlusions and suffer from reduced reliability due to a low signal-to-noise ratio, particularly over long distances [**?**]. Variability in lighting, weather conditions and the physical state of the refuelling adapter further complicates detection and localisation [**?**]. Vision-based systems must operate in real time and with minimal computational load, adding a new layer of complexity.

The development of high-quality data sets for training and monitoring automated refuelling systems is another major challenge. Collecting and processing these datasets is often time-consuming and requires meticulous configuration [**?**].

Visual measurement methods based on artificial features, such as spray marks or LEDs, are commonly used in today's automated refuelling systems. The VisNav system, developed by Valsek, uses light-emitting diodes emitting at different frequencies to locate the centre of a beacon, employing a Gaussian least squares differential correction algorithm to calculate the position of the [**?**] refuelling adaptor. Despite the advantages of these methods, they remain sensitive to occlusion and low signal-to-noise ratios at long distances.

Recent advances in Automatic Aircraft Ground Refuelling (AAGR) rely on computer vision, artificial intelligence and robotics to achieve high levels of automation. The integration of these technologies with Big Data has significantly improved the feasibility and accuracy of AAGR systems thanks to the creation of large datasets for training and validation. For example, an Aircraft Ground Refuelling (AGR) dataset comprising more than 3,000 images from 13 databases was expanded to more than 26,000 images after augmentation, enabling AGR scene recognition through image mining, augmentation and classification [**?**]. In addition, recent innovations have introduced hybrid datasets combining real and synthetic data for training and validating [**?**] systems. This approach offers a wide range of scenarios and conditions, improving the robustness and accuracy of automated refuelling systems.

Furthermore, in the field of autonomous aerial refuelling, current technologies rely mainly on vision-based systems and sophisticated control algorithms. These methods use a variety of sensors, including monocular and binocular cameras, to detect and track drugs and refuelling probes. **?**] demonstrated the feasibility of real-time drug recognition and 3D localisation for autonomous aerial drone refuelling using monocular computer vision. In addition, **?**] explored the application of 3D flash lidar for drug tracking [**?**]. The probe and drogue refuelling system involves the refuelling aircraft towing a refuelling hose with a drogue at the end, while the pilot of the receiving aircraft manoeuvres to insert the probe into the drogue. For unmanned aerial refuelling, this process is carried out autonomously. Although DGPS offers high location accuracy, it faces challenges such as lock-in problems and low bandwidth in air-to-air refuelling applications [**?**].

2.2 Object Detection and Tracking in Computer Vision

In Computer Vision, Object Detection refers to the identification and location of individual objects within an image, providing both spatial information (bounding boxes) and confidence scores, which represent the probability that each detected object belongs to the predicted class [**?**]. For example, in the following image, there are five detections, including one 'ball' with a confidence level of 98% and four 'people' with confidence levels of 98%, 95%, 97% and 97%.

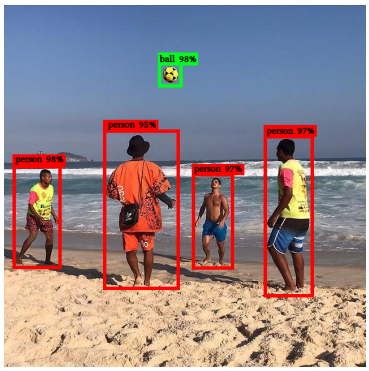

Figure 2.1: Example of outputs from an object detector [**?**].

Over the last few decades, Object Detection models based on Deep Learning have enjoyed remarkable success. These models fall into two main categories: two-stage detectors and single-stage detectors.

On the one hand, two-stage detectors, such as R-CNN [**?**], Fast R-CNN [**?**], Faster R-CNN [**?**] and R-FCN [**?**], first generate region proposals and then refine these proposals into precise anchor boxes. While these models excel in detection accuracy, they typically suffer from large model sizes and slower detection speeds [**? ?**].

On the other hand, single-stage detectors, including the SSD (Single Shot Multibox Detector) [**?**], YOLO (You Only Look Once) series [**? ? ? ? ? ? ? ? ? ? ? ? ? ?**], and RetinaNet [**?**] directly predict object locations and categories in a single network pass. These models are known for their high detection speeds but sometimes compromise accuracy [**? ?**].

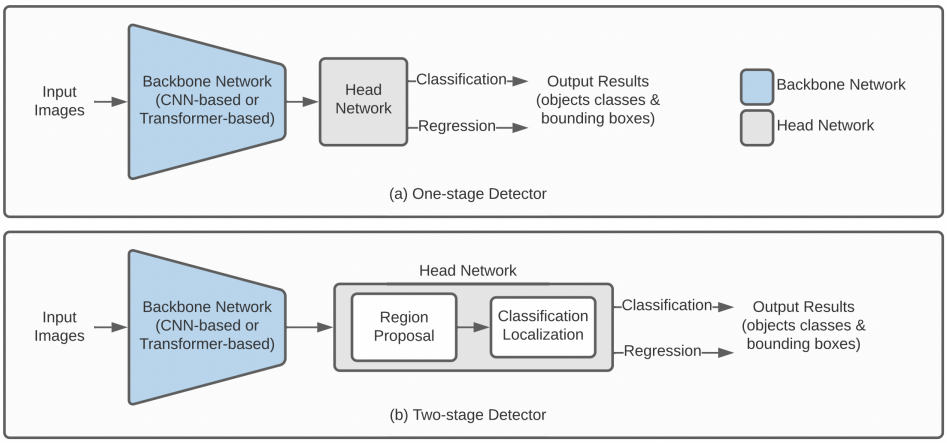

Figure 2.2: Basic deep learning-based one-stage vs two-stage object detection model architectures [**?**].

Evaluating Object Detection models involves several key metrics to measure their performance. One common metric is Intersection over Union (IoU), which measures the overlap between a predicted bounding box and a ground-truth bounding box, as shown in Figure 2.3.

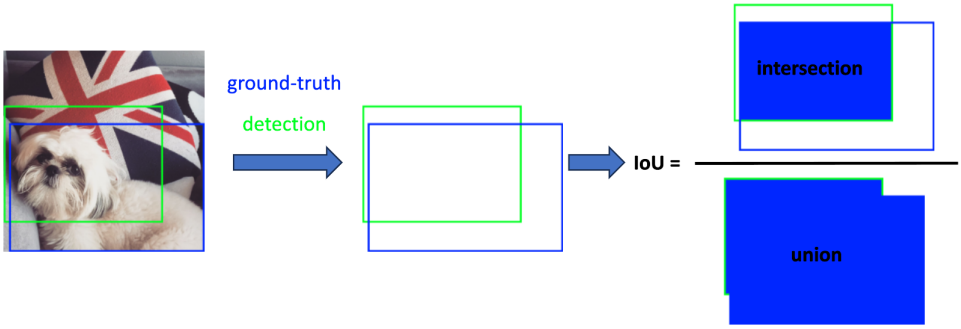

Figure 2.3: Intersection over Union (IoU) between a detection (in green) and ground-truth (in blue). [**?**]

Based on the IoU metric, a detection can be classified as a **True Positive (TP)** or a **False Positive (FP)** depending on whether the IoU value exceeds a certain threshold (T_{IoU}). If the IoU is above the threshold, the detection is considered correct (TP); otherwise, it is classified as a False Positive (FP). Additionally, **False Negatives (FN)** refer to ground truth objects not detected by the model, while **True Negatives (TN)** are correctly classified background detections [**?**]. These classifications allow for the calculation of the following metrics:

- **Precision**: The ratio of True Positives to the total number of detections, measuring the model's ability to avoid false positives:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.1}$$

- **Recall**: The ratio of True Positives to the total number of ground-truth objects, measuring the model's ability to avoid false negatives:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.2}$$

Common metrics used to evaluate Object Detection include:

- **Average Precision (AP)**: This combines precision and recall, providing a single figure summarizing the model's performance across different confidence thresholds. Common versions are AP@.5 (with a threshold of 0.5 IoU) and AP@[.5:.05:.95], which calculates the average of AP values over several IoU thresholds [**?**].
- **Average Recall (AR)**: This measures the recall of the model averaged across multiple IoU thresholds. It can be computed for different numbers of detections per image, such as AR@1, AR@10, etc. [**?**].
- **Inference Time**: The time taken by the model to process an image, which is critical for applications requiring real-time detection [**?**].
- **Model Size**: The number of parameters or the size of the model, affecting deployment, especially on devices with limited resources [**?**].

- **Efficiency**: This considers the trade-off between accuracy and speed, often visualized using the AP vs. inference time curve [**?**].

In addition to Object Detection, Object Tracking is another critical task in Computer Vision, involving the continuous monitoring of objects across video frames. Object Tracking methods can be broadly classified into two categories: **Generative Trackers** and **Discriminative Trackers** [**?**]. Generative trackers are capable of handling challenging scenarios such as occlusion and large-scale variation through particle sampling strategies, often integrated with various appearance models, including sparse representation and energy of motion. Discriminative trackers, by contrast, build robust classifiers using hand-crafted or deep features [**?**]. The combination of generative and discriminative approaches, as well as the integration of deep learning techniques such as fully revolutionary networks and Transformer models, has led to significant improvements in object detection performance [**?** **?** **?**]. In addition, the speed and computational requirements of these algorithms are critical factors influencing their practical applicability [**?** **?** **?**]. Advanced techniques in object tracking leverage both generative and discriminative models to amplify tracking efficacy. The utilisation of deep trackers has evidenced superior results on public tracking datasets, attributed to their potent feature extractors, accurate bounding box regressors, and discriminative classifiers [**?**]. Techniques such as deformable convolution and Transformer models extend traditional convolution or correlation methodologies to execute global feature matching, thereby enhancing tracking accuracy. The incorporation of contextual or knowledge information can substantially elevate performance, with methodologies like Particle Filtering, also recognised as Sequential Monte Carlo (SMC) methods, framed as problems of Bayesian inference in state space [**?** **?**]. The extended Kalman Filtering (EKF) is another advanced technique that has been employed to improve tracking accuracy by predicting the current status through the previous status and modifying the prediction result based on observation information [**?** **?**]. Despite these advancements, the integration of these methods in a complementary manner remains an open research area with substantial potential for advancing the field [**?** **?**].

2.3 Deep Learning for Spacio-Temporal Prediction

Time series prediction involves processing sequential data to predict future events or values. Various deep learning models have been applied to this task, requiring several preparatory steps such as collecting data, designating attribute types, dealing with inconsistencies and storing datasets. These datasets are usually classified into units of time such as seconds, minutes and hours, allowing the construction of metadata for machine learning [**?**].

Early Approaches and Models

The problem of predicting the future locations of objects has been extensively studied, particularly for static surveillance cameras. Initial efforts utilised recurrent neural networks (RNNs), including long-term memory networks (LSTMs) and gated recurrent units (GRUs), in an encoder-decoder format to encode past observations and decode future locations.

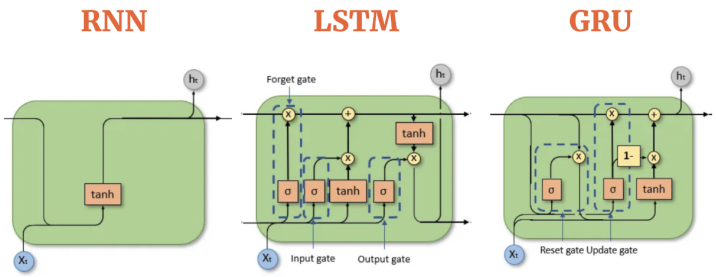

Figure 2.4: Comparing different Sequence models: RNN, LSTM, and GRU. Source: Colah's blog. Compiled by AIML.com

Early models integrated additional inputs such as environmental data and semantic actions to enhance prediction accuracy. For instance, **?**] proposed a Social-LSTM to model pedestrian trajectories and interactions, further improving global context capture through a social pooling module.

Transformer Models for Frame-Based Prediction

Transformer model, introduced by **?**], has revolutionised sequence modeling with its efficient and powerful network structure. For video prediction, transformers leverage self-attention mechanisms to capture long-range dependencies across frames. The **?**] study shows how transformers can effectively learn mobility patterns from historical frame sequences, achieving state-of-the-art performance in next-location prediction tasks.

Frame-Based Approaches in Spatio-Temporal Prediction

Video prediction, a critical task in spatio-temporal data analysis, involves forecasting future frames based on past and current frames. This process requires understanding both spatial and temporal dynamics within the data.

Convolutional LSTM (ConvLSTM)

ConvLSTM integrates convolutional operations into LSTM units to better capture spatial features in addition to temporal dependencies [**?**]. By processing video data as sequences of frames, ConvLSTM effectively models the spatio-temporal dependencies necessary for accurate video prediction. Each frame serves as a spatial unit, and the sequence of frames provides temporal context, allowing the model to predict future frames by learning from the patterns in previous ones.

Cubic LSTM for Video Prediction

The Cubic Long Short-Term Memory (CubicLSTM) unit, as proposed by **?**], extends the capabilities of ConvLSTM by separately processing spatial and temporal information through three branches: temporal, spatial, and output. This approach mitigates the computational burden and enhances prediction accuracy. Specifically:

- The *temporal branch* processes motion information by analysing the sequence of frames over time.

- The *spatial branch* captures object information within individual frames.
- The *output branch* combines temporal and spatial information to generate predicted future frames.

This separation of concerns allows the model to handle the complexities of video data more effectively, resulting in more accurate predictions of future frames.

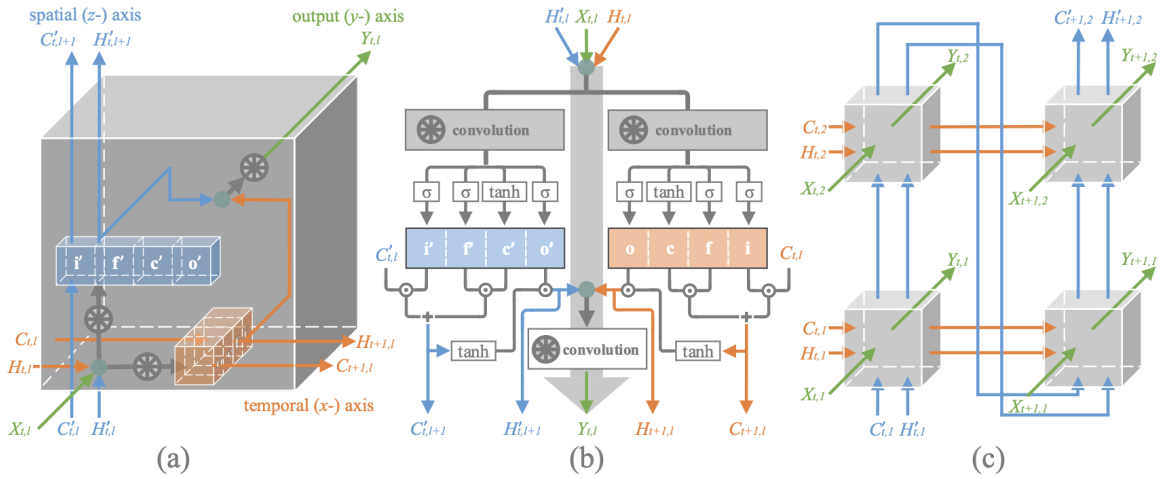

Figure 2.5: Cubic LSTM Architecture. (a) 3D structure of the CubicLSTM unit. (b) Topological diagram of the CubicLSTM unit. (c) Two-spatial-layer RNN composed of CubicLSTM units. The unit consists of three branches, a spatial (z-) branch for extracting and recognizing moving objects, a temporal (y-) branch for capturing and predicting motions, and an output (x-) branch for combining the first two branches to generate the predicted frames. Source: **?**]

Unsupervised Video Forecasting with Flow Parsing

?] introduced an unsupervised video forecasting approach that incorporates a flow parsing mechanism. This model separates the motion and appearance learning processes:

- The *motion stream* predicts optical flow between frames to capture dynamic changes.
- The *appearance stream* reconstructs frames to preserve spatial details.

By integrating these streams, the model can predict future frames with improved motion accuracy and appearance fidelity.

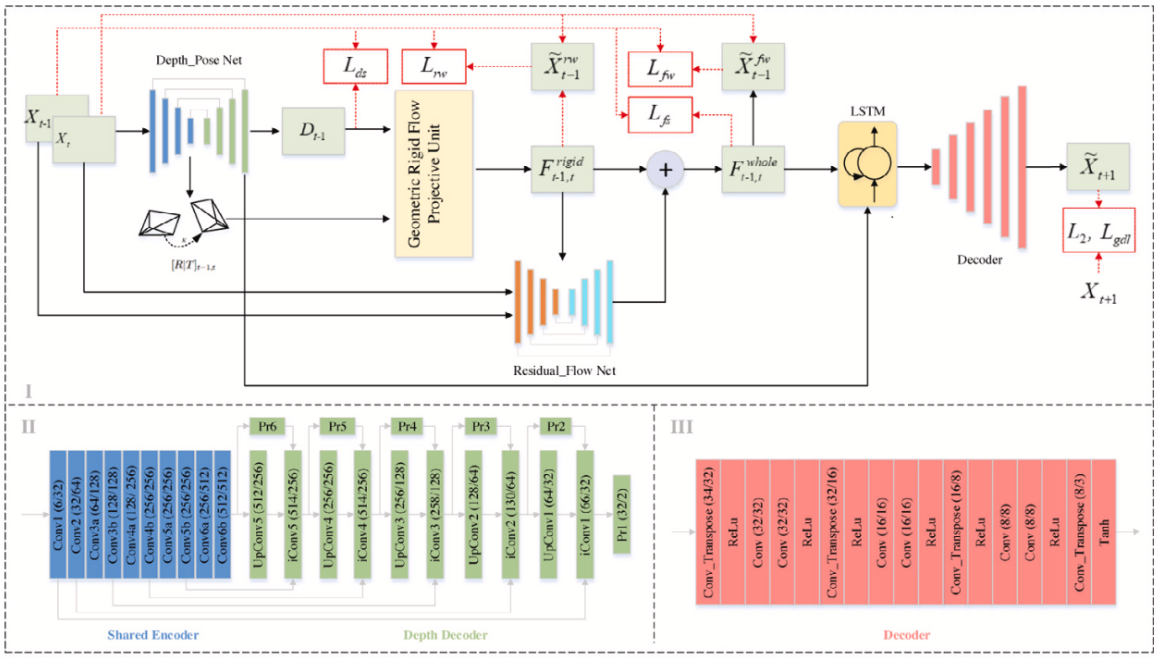

Figure 2.6: Model Architecture. Source: **?**]

Joint Optimization with Synthesis and Optical Flow Estimation

?] proposed a method that jointly optimizes frame synthesis and optical flow estimation. Their approach leverages:

- A *frame synthesis network* to generate predicted frames.
- An *optical flow estimation network* to capture motion dynamics between frames.

This joint optimization enhances the model's ability to produce accurate and visually coherent future frames.

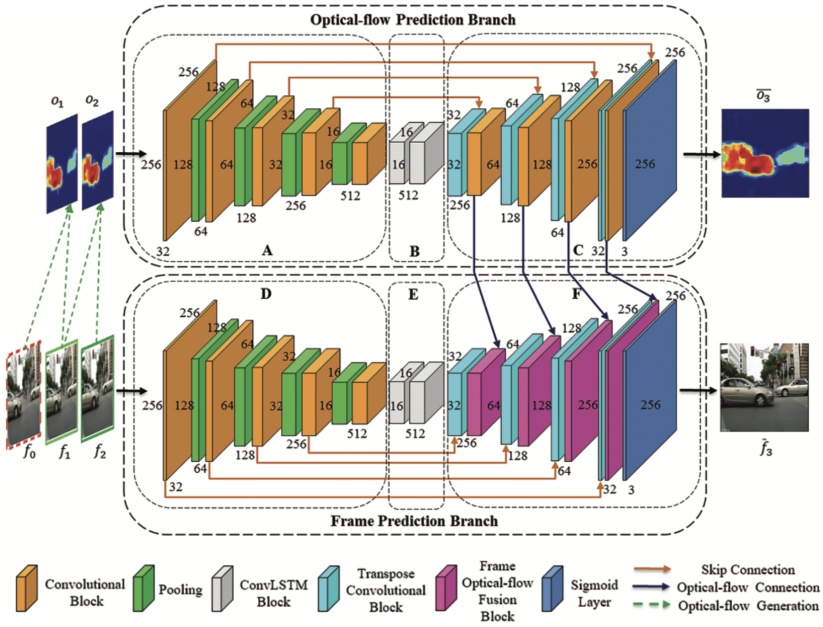

Figure 2.7: FPNet-OF model architecture. Source: **?**]

Fusion-GRU

?] developed the Fusion-Gated Recurrent Unit (Fusion-GRU) model to predict the future bounding boxes of traffic agents in risky driving scenarios. This model leverages multiple sources of information, including location-scale data, monocular depth information, and optical flow data, to capture complex interactions among information cues and transform them into hidden representations. The Fusion-GRU model uses an intermediary estimator and self-attention aggregation layer to enhance sequential dependencies for long-term predictions, demonstrating superior performance in challenging driving scenarios.

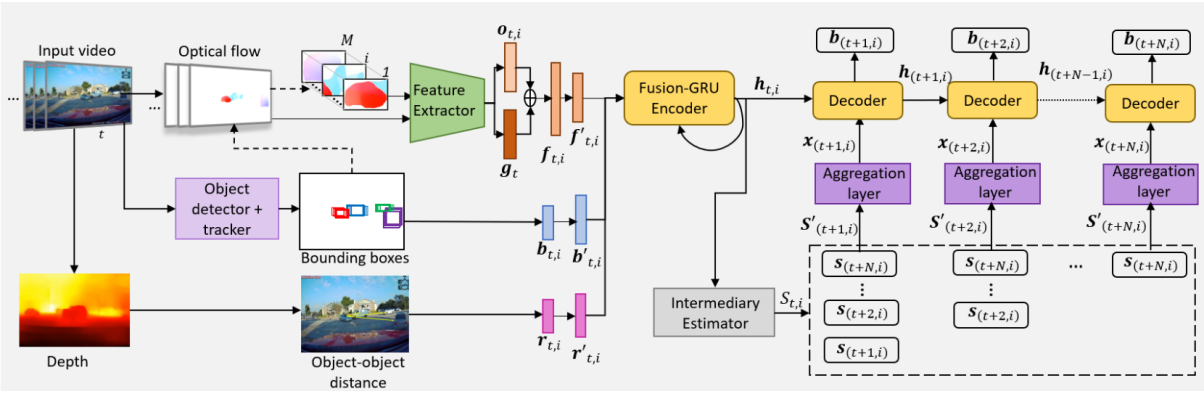

Figure 2.8: Fusion-GRU model architecture. Source: **?**]

Dual-Branch Spatial-Temporal Learning Network

Huang and Guan [**?**] proposed a dual-branch video prediction network that aims to generate high-quality future frames by simultaneously capturing complex motion patterns and preserving appearance information. Their network includes two distinct units:

- The *motion prediction unit (MPU)* focuses on inter-frame motion and intra-frame appearance by using depth and multiple-scale convolutions, along with temporal attention mechanisms to enhance feature interactions over time.
- The *spatial prediction unit (SPU)* concentrates on spatial information, ensuring appearance consistency across video frames by capturing various appearance features.

This dual-branch approach addresses the limitations of relying on external information or complex state transition units, resulting in better visual quality and fewer blurry artifacts in predicted frames.

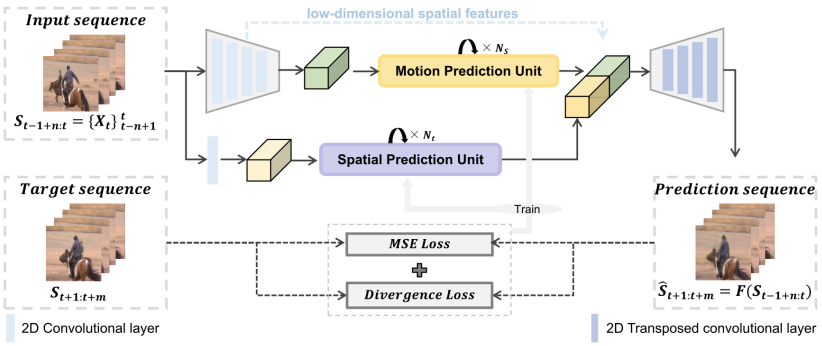

Figure 2.9: Dual-Branch Spatial-Temporal Learning Network. Source: **?**]

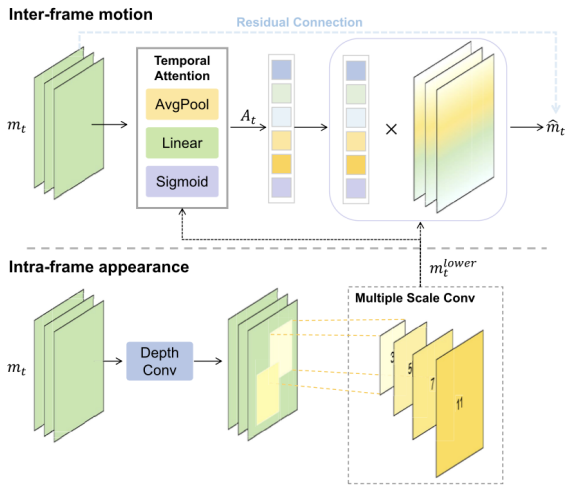

Figure 2.10: Dual-Branch Spatial-Temporal Learning Network. Source: **?**]

Applications and Case Studies

In various applications, the frame-based approach to spatio-temporal prediction has proven highly effective:

Moving-MNIST Dataset

The Moving-MNIST dataset consists of sequences of moving handwritten digits. Each sequence includes a series of frames, and the goal is to predict future frames based on the observed ones. This dataset contains 10000 videos, each consisting of 20 frames [**?**]. The CubicLSTM-based CubicRNN demonstrated superior accuracy compared to traditional ConvLSTM models, showcasing its ability to capture both motion and spatial features effectively [**?**].

Figure 2.11: 2-digit Moving MNIST data by **?**]

Robotic Pushing Dataset

This dataset involves sequences of robotic arms pushing objects, with each frame representing a step in the sequence. The dataset contains 3456 training images with labels and 1024 validation images with labels [**?**]. The CubicLSTM model, combined with convolutional

dynamic neural advection (CDNA) models, achieved higher accuracy and generated clearer frames compared to ConvLSTM and CNN-based models. This application highlights the importance of accurately predicting future frames to understand and anticipate the movements of robotic systems [**?**].

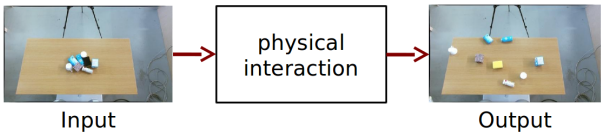

Figure 2.12: Robotic Pushing Dataset. Source: **?**]

KTH Action Dataset

The KTH Action dataset features videos of people performing various actions, with each video divided into frames. The dataset contains 2391 sequences [**?**]. The CubicLSTM model provided more accurate and visually consistent predictions of future frames compared to other state-of-the-art models like DrNet and MCnet. This demonstrates the model's effectiveness in understanding and predicting human actions over time [**?**].

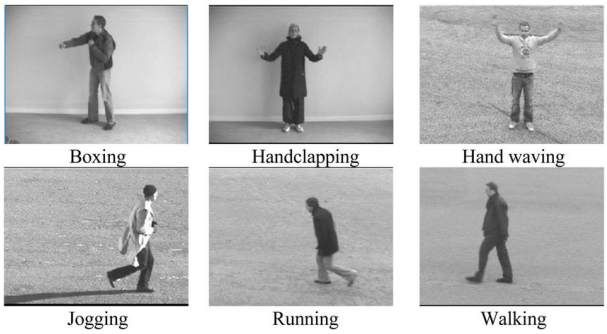

Figure 2.13: KTH Action Dataset. Source: **?**]

UCF Sports Dataset

The UCF Sports dataset consists of a collection of footage collected from various sports that are typically shown on television channels such as the BBC and ESPN. It comprises a total of 150 sequences [**?**]. Huang and Guan's dual-branch network was tested on the UCF Sports dataset, which includes complex motion patterns and large movements. Their method outperformed other state-of-the-art methods in terms of visual quality and motion accuracy, as evidenced by superior metrics such as PSNR, SSIM, and LPIPS. The results validate the network's ability to handle intricate video prediction tasks without relying on additional information [**?**].

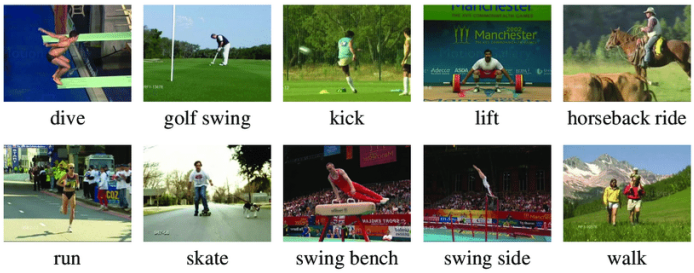

Figure 2.14: UCF Sports Dataset. Source: **?**]

Chapter 3

Methodology

3.1 Dataset Configuration

3.1.1 Provided Dataset Description

The 'Indoor Hangar Fueling Port Detection and Tracking Dataset' (HARD) (HARD) is an integral part of Phase-1 of the ONEHeart project, funded by UKRI. The ONEHeart project aims to develop an automated aircraft refuelling system based on computer vision and robotics technology. This dataset can be utilised for various purposes, including but not limited to fueling port detection, fueling port tracking, camera pose estimation, and visual image processing. It is provided under the terms of the UKRI funding agreement. Unauthorised use, distribution, or reproduction is prohibited.

The HARD dataset consists of 21 video sequences captured in an indoor hangar at the Aerospace Integration Research Centre (AIRC). The videos were recorded using an Intel RealSense D435 sensor, which provides depth information in addition to RGB data. The target area for detection and tracking is near the fueling port of an Airbus A320 wing. The videos were recorded under different lighting conditions, including indoor artificial lighting and natural daylight, to simulate real-world scenarios.

There are three states of the fueling port in the dataset:

- **CLOSED**: The fueling port is closed.
- **OPEN**: The fueling port is open.
- **SEMI-OPEN**: The fueling port is partially open.

3.1.2 Data Annotation

The HARD dataset provided for this project was not fully annotated. Therefore, the first step in the data preparation process was to annotate the dataset. This annotation process was crucial to enable the training of machine learning models for accurate fueling port detection and tracking.

The annotation process involved several steps:

1. **Initial Manual Annotation**: Initially, 100 frames from each video sequence were manually annotated. This involved labeling the fueling port in each frame, which required identifying and marking the exact location of the fueling port using bounding boxes. This

was done using the Label Studio tool, a powerful annotation platform that allows users to create and manage annotations in images and videos.

2. **Tool Used - Label Studio**: Label Studio was chosen for its flexibility and ease of use. It supports various annotation formats and integrates well with machine learning workflows. Users can draw bounding boxes around objects of interest, in this case, the fueling port, to create labeled datasets.

3. **Preliminary Dataset Creation**: The initial manual annotations created a preliminary dataset. This dataset was used to train a YOLOv10 (You Only Look Once, version 10) model for fueling port detection.

4. **Model Training**: The annotated dataset was used to train the YOLOv10 model. The model learned to detect the fueling port from the annotated images, improving its accuracy with each training iteration.

5. **Automated Annotation**: After training the YOLOv10 model, it was implemented as a backend service for Label Studio. This was deployed as a Docker container, allowing the model to be used for automated annotation of the remaining images in the dataset. The model predicted the location of the fueling port in new frames, and these predictions were used to annotate the rest of the dataset automatically.

6. **Review and Quality Control**: Each automatically generated annotation was reviewed manually to ensure accuracy. This review process involved verifying the location of the fueling port in each frame and making necessary corrections to the annotations. This thorough quality control ensured that the entire dataset was consistently and accurately labeled.

This comprehensive annotation process ensured that the entire dataset was accurately labeled, providing a robust foundation for training machine learning models aimed at fueling port detection and tracking. The combination of manual and automated annotation techniques maximised efficiency while maintaining high annotation quality.

3.1.3 Summary of Available Videos

Each video was assigned to a specific dataset (train, val, and test), with an approximate split of 70% for training, 15% for validation, and 15% for testing. The following table presents the available videos in the dataset along with the number of frames for each video and their assignment:

Type	Video Name	Number of Frames	Assignment
Closed	**video_lab_platform_1**	**624**	**train**
Closed	video_lab_platform_2	639	train
Closed	video_lab_platform_5	398	train
Closed	video_lab_platform_7	412	train
Closed	video_lab_platform_8	470	train
Closed	video_lab_platform_9	373	train
Closed	video_lab_manual_1	746	train
Closed	video_lab_platform_3	569	val
Closed	video_lab_platform_4	247	val
Closed	video_lab_platform_6	303	test
Closed	test_outdoor1	499	test
Open	**video_lab_open_1_____1**	**497**	**train**
Open	video_lab_open_1_____2	602	train
Open	video_lab_open_1_____3	313	train
Open	video_lab_open_1_____4	310	train
Open	test_indoor2	310	val
Open	test_indoor1	314	test
Semi-Open	**video_lab_semiopen_1_____1**	**739**	**train**
Semi-Open	video_lab_semiopen_1_____2	439	train
Semi-Open	video_lab_semiopen_1_____4	372	val
Semi-Open	video_lab_semiopen_1_____3	383	test

Table 3.1: Summary of available videos in the HARD dataset with their assignment.

3.1.4 Initial Data Distribution

Before balancing the data, the distribution of video frames across the different datasets (train, validation, and test) for each state of the fueling port (CLOSED, OPEN, and SEMI-OPEN) was as follows:

Type	Total Frames	Train	Test	Validation
CLOSED	5280	3662 (69.36%)	802 (15.19%)	816 (15.45%)
OPEN	2346	1722 (73.40%)	314 (13.38%)	310 (13.21%)
SEMI-OPEN	1933	1178 (60.94%)	383 (19.81%)	372 (19.24%)

Table 3.2: Distribution of frames across train, test, and validation sets for each state in the HARD dataset before balancing.

As shown in 3.2, the dataset initially had an imbalance in the number of frames for each state. For instance, the CLOSED state had significantly more frames compared to the OPEN and SEMI-OPEN states. This imbalance could lead to biased training and inaccurate model performance, as the model might become overly familiar with the more prevalent CLOSED state and underperform on the less represented states.

3.1.5 Balanced Data Distribution

To address this issue, a balancing strategy was employed to create a more uniform dataset. The steps taken were as follows:

1. **Shuffling and Subsetting**: Each state (CLOSED, OPEN, and SEMI-OPEN) subset was shuffled, and only the minimum number of frames from each state was kept. This step ensured that the number of frames for each state was equal, avoiding bias toward any particular state.

2. **Merging Subsets**: The subsets were merged to create three balanced datasets: Training, Validation, and Test, each with an equal number of frames for each state.

3. **Final Shuffling and Resizing**: Each balanced dataset was shuffled again and resized to maintain the required split ratio of 70% for training, 15% for validation, and 15% for testing.

The balanced dataset will be used for training and evaluating the object detection model, while the full dataset will be utilised for sequence model training and framework evaluation.

The resulting distribution of frames across the train, test, and validation sets for each state after balancing is as follows:

Dataset	Total Frames
Train	3534 (69.57%)
Test	773 (15.22%)
Val	773 (15.22%)

Table 3.3: Distribution of frames across train, test, and validation sets for each state in the HARD dataset after balancing.

The primary reason for balancing the dataset is to ensure that the object detection model is equally trained on all states of the fueling port. An imbalanced dataset could cause the model to perform well on the more common states (like CLOSED) while underperforming on the less common states (like OPEN and SEMI-OPEN). By balancing the dataset:

- **Improved Generalization**: The model will have an equal opportunity to learn from all states, improving its generalization and robustness.

- **Avoiding Bias**: Equal representation of each state prevents the model from becoming biased towards any particular state.

- **Consistent Performance**: This strategy ensures consistent performance across all states, which is critical for the reliable operation of the automated refueling system.

3.1.6 Example Images from the Dataset

The following figures show annotated examples of the fueling port in different states:

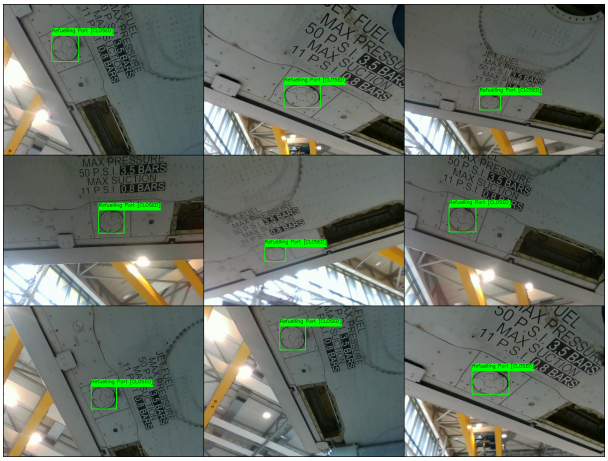

Figure 3.1: Annotated images of the fueling port in the CLOSED state.

Figure 3.2: Annotated images of the fueling port in the OPEN state.

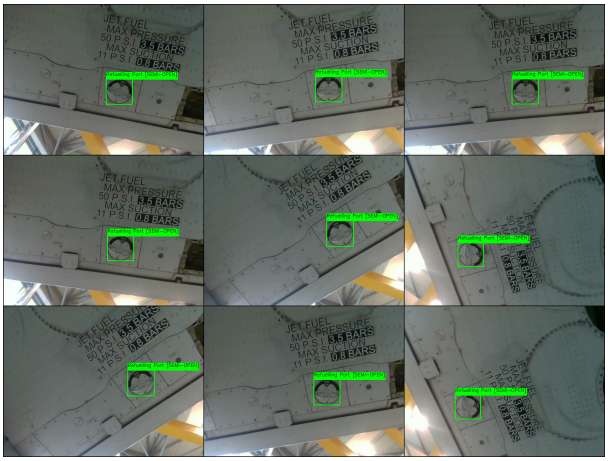

Figure 3.3: Annotated images of the fueling port in the SEMI-OPEN state.

3.2 Framework Design

3.3 Model Design

3.3.1 LSTM-based Sequence Model

The framework for predicting future positions and velocities of objects in a video stream leverages an encoder-attention-decoder architecture. This design captures temporal dependencies and spatial relationships effectively. The overall architecture consists of four main components: two encoders, the attention mechanism, and two decoders. Each component is designed to handle specific aspects of the sequence modeling task, from encoding input sequences to generating context-aware predictions

3.3.1.1 Input Representation

The model takes as input sequences of bounding boxes and their velocities extracted from video frames. Each bounding box is represented by its center coordinates $(x_{\text{center}}, y_{\text{center}})$, width (w), and height (h). Corresponding velocities are represented by changes in these values: $(\Delta x, \Delta y, \Delta w, \Delta h)$.

Thus, each input vector $\mathbf{p}_t$ and $\mathbf{v}_t$ at time t can be represented as:

$$\mathbf{p}_t = (x_{\text{center},t}, y_{\text{center}}, w_t, h_t) \tag{3.1}$$

$$\mathbf{v}_t = (\Delta x_t, \Delta y_t, \Delta w_t, \Delta h_t) \tag{3.2}$$

These sequences are fed into the encoders to extract meaningful temporal features.

3.3.1.2 Encoders

The framework employs two separate LSTM encoders: one for processing the sequence of bounding boxes and another for processing the sequence of velocities. Each encoder is designed to capture the temporal dependencies within its respective input data.

The bounding box encoder is defined as:

$$H_p, (h_{p_n}, c_{p_n}) = \text{LSTM}_p(\mathbf{P}) \tag{3.3}$$

where $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_T)$ is the input sequence of bounding boxes, $H_p = (h_{p_1}, h_{p_2}, \ldots, h_{p_T})$ represents the sequence of hidden states, and (h_{p_n}, c_{p_n}) are the final hidden and cell states of the bounding box encoder.

The velocity encoder is defined as:

$$H_v, (h_{v_n}, c_{v_n}) = \text{LSTM}_v(\mathbf{V}) \tag{3.4}$$

where $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_T)$ is the input sequence of velocities, $H_v = (h_{v_1}, h_{v_2}, \ldots, h_{v_T})$ represents the sequence of hidden states, and (h_{v_n}, c_{v_n}) are the final hidden and cell states of the velocity encoder.

3.3.1.3 Attention Mechanism

The attention mechanism is crucial for allowing the model to focus on relevant parts of the input sequences when making predictions. It computes a set of attention weights that highlight important hidden states from both encoders. These weights are used to form context vectors, which provide the decoders with relevant information from the input sequences.

The attention weights α_t are calculated as follows:

$$\alpha_t = \text{softmax}(v^\top \tanh(W_{h_p} H_p + W_{h_v} H_v + W_s s_{t-1})) \tag{3.5}$$

where W_{h_p} and W_{h_v} are learned weight matrices for the bounding box and velocity hidden states, respectively, W_s is a learned weight matrix for the previous decoder state, and v is a learned vector. The context vectors c_{p_t} and c_{v_t} are then computed as:

$$c_{p_t} = \sum_{i=1}^{T} \alpha_{t,i} h_{p_i} \tag{3.6}$$

$$c_{v_t} = \sum_{i=1}^{T} \alpha_{t,i} h_{v_i} \tag{3.7}$$

These context vectors are passed to the decoders to generate the next time step predictions.

3.3.2 Decoders

The framework employs two separate LSTM decoders: one for generating predictions for future bounding boxes and another for generating predictions for future velocities. Each decoder takes as input the previous output and the context vectors from the attention mechanism.

The bounding box decoder's input at each time step t is the concatenated vector of the previous output and the context vector from the bounding box encoder:

$$\mathbf{d}_{p_t} = [\mathbf{p}_{t-1}, c_{p_{t-1}}] \tag{3.8}$$

The LSTM in the bounding box decoder processes this input to update its hidden and cell states:

$$s_{p_t}, (h_{p_t}, c_{p_t}) = \text{LSTM}_p(\mathbf{d}_{p_t}, (h_{p_{t-1}}, c_{p_{t-1}})) \tag{3.9}$$

The final output for the bounding box is obtained through a fully connected layer with a sigmoid activation function to ensure the output values are between 0 and 1:

$$\mathbf{p}_t = \text{Sigmoid}(\text{Linear}_p(s_{p_t})) \tag{3.10}$$

Similarly, the velocity decoder's input at each time step t is the concatenated vector of the previous output and the context vector from the velocity encoder:

$$\mathbf{d}_{v_t} = [\mathbf{v}_{t-1}, c_{v_{t-1}}] \tag{3.11}$$

The LSTM in the velocity decoder processes this input to update its hidden and cell states:

$$s_{v_t}, (h_{v_t}, c_{v_t}) = \text{LSTM}_v(\mathbf{d}_{v_t}, (h_{v_{t-1}}, c_{v_{t-1}})) \tag{3.12}$$

The final output for the velocity is obtained through a fully connected layer without any activation function. This design choice allows the output values to range freely, including negative values, which is necessary because velocities can be positive or negative:

$$\mathbf{v}_t = \text{Linear}_v(s_{v_t}) \tag{3.13}$$

The decoders continue this process for the entire prediction horizon, using their own previous outputs as inputs for future predictions.

3.3.3 Transformer-Based Architecture

The proposed model leverages a Transformer architecture to predict future positions and velocities of objects in a video stream. Transformers, introduced by Vaswani et al. (2017), have demonstrated superior performance in sequence-to-sequence tasks by effectively capturing long-range dependencies through self-attention mechanisms. Our architecture consists of three main components: an encoder, a decoder, and a multi-head self-attention mechanism. Each component is designed to handle specific aspects of the sequence modeling task, from encoding input sequences to generating context-aware predictions.

3.3.3.1 Input Representation

The model takes as input sequences of bounding boxes and their velocities extracted from video frames. Each bounding box is represented by its center coordinates $(x_{\text{center}}, y_{\text{center}})$, width (w), and height (h). Corresponding velocities are represented by changes in these values: $(\Delta x, \Delta y, \Delta w, \Delta h)$. Thus, each input vector $\mathbf{p}_t$ and $\mathbf{v}_t$ at time t can be represented as:

$$\mathbf{p}_t = (x_{\text{center},t}, y_{\text{center},t}, w_t, h_t) \tag{3.14}$$

$$\mathbf{v}_t = (\Delta x_t, \Delta y_t, \Delta w_t, \Delta h_t) \tag{3.15}$$

These sequences are fed into the encoder to extract meaningful temporal features.

3.3.3.2 Encoder

The encoder consists of multiple layers, each containing two main components: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The input to the encoder is a sequence of bounding box and velocity vectors. The encoder processes these sequences independently using the following steps:

- **Positional Encoding**: Since the Transformer model does not have a built-in notion of order, positional encodings are added to the input embeddings to incorporate sequence order information.
- **Self-Attention**: The multi-head self-attention mechanism allows the model to focus on different parts of the input sequence simultaneously, capturing dependencies regardless of their distance in the sequence.
- **Feed-Forward Network**: A position-wise fully connected feed-forward network is applied to the output of the self-attention mechanism to introduce non-linearity and further process the information.

- **Residual Connection and Layer Normalization**: Residual connections around each sub-layer followed by layer normalization help in stabilizing the training and improving gradient flow.

3.3.3.3 Decoder

The decoder is similarly composed of multiple layers, with each layer containing three main components: a multi-head self-attention mechanism, a multi-head encoder-decoder attention mechanism, and a position-wise fully connected feed-forward network. The decoder generates predictions for future bounding boxes and velocities using the following steps:

- **Masked Self-Attention**: The decoder's self-attention mechanism is masked to prevent positions from attending to subsequent positions, ensuring that the prediction for a position depends only on known outputs.
- **Encoder-Decoder Attention**: This mechanism allows the decoder to focus on relevant parts of the encoder's output, providing context from the input sequence.
- **Feed-Forward Network**: Similar to the encoder, a fully connected feed-forward network is applied to the output of the attention mechanisms.
- **Residual Connection and Layer Normalization**: As in the encoder, residual connections and layer normalization are applied to stabilize training and improve gradient flow.

3.3.3.4 Multi-Head Self-Attention

The multi-head self-attention mechanism is a key component of the Transformer architecture, enabling the model to jointly attend to information from different representation subspaces. It works by projecting the input sequence into multiple sets of queries, keys, and values, performing scaled dot-product attention for each set, and concatenating the results. This process allows the model to capture various aspects of the input sequence, enhancing its ability to learn complex patterns.

3.3.3.5 Output Generation

The output of the decoder is passed through a linear layer to generate predictions for the bounding box coordinates and velocities. Separate linear layers are used for bounding box predictions (with a sigmoid activation to ensure values are between 0 and 1) and velocity predictions (without activation to allow for positive and negative values).

3.3.3.6 Loss Function and Training

The model is trained to minimize the mean squared error between the predicted and actual bounding boxes and velocities. The loss function is defined as:

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\left[(x_{i,t}-\hat{x}_{i,t})^2+(y_{i,t}-\hat{y}_{i,t})^2+(w_{i,t}-\hat{w}_{i,t})^2+(h_{i,t}-\hat{h}_{i,t})^2+(\Delta x_{i,t}-\hat{\Delta x}_{i,t})^2+(\Delta y_{i,t}-\hat{\Delta y}_{i,t})^2+(\right. \tag{3.16}$$

The Adam optimizer is used to update the model parameters, and early stopping based on validation loss is employed to prevent overfitting and ensure generalization.